

# Android Programming

1. Implementing a User Interface
2. Storing, Retrieving and Exposing Data

# Implementing a User Interface in Android

MMI2



# Outline

- Introduction
- Programmatic vs. XML Layout
- Common Layout Objects
- Hooking into a Screen Element
- Listening for UI Notifications
- Applying a Theme to Your Application



# Introduction

Implementing a User Interface


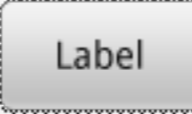





# Introduction

- **Activity**
  - Basic functional unit of an Android application
  - But by itself, it does not have any presence on the screen
- **Views and Viewgroups**
  - Basic units of user interface expression on the Android platform

# Beautiful View from up here...

- android.view.View
  - Stores layout and content for a specific rectangular area of the screen
  - Handles measuring and layout, drawing, focus change, scrolling, and key/gestures
  - Base class for widgets

- Text 
- EditText
- InputMethod
- MovementMethod
- Button 
- RadioButton
- Checkbox  
- ScrollView 

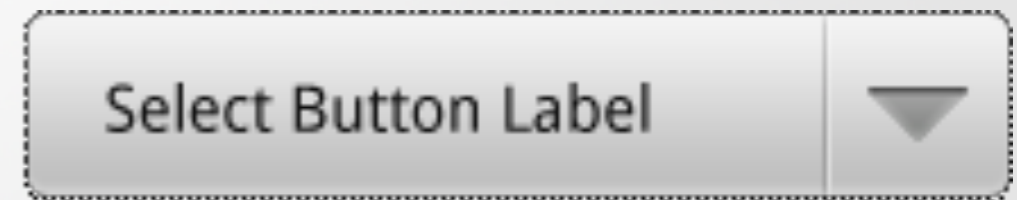
```
package com.android.hello;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        tv.setText("Hello, Android");
        setContentView(tv);
    }
}
```

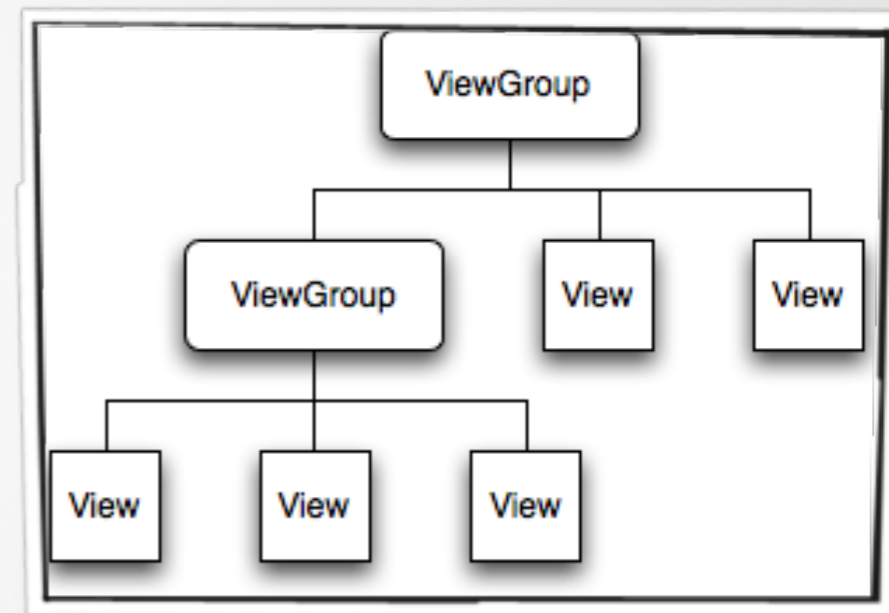
# Viewgroups

- `android.view.ViewGroup`
  - Contains and manages a subordinate set of views and other viewgroups
  - Base class for layouts



# Tree-Structured UI

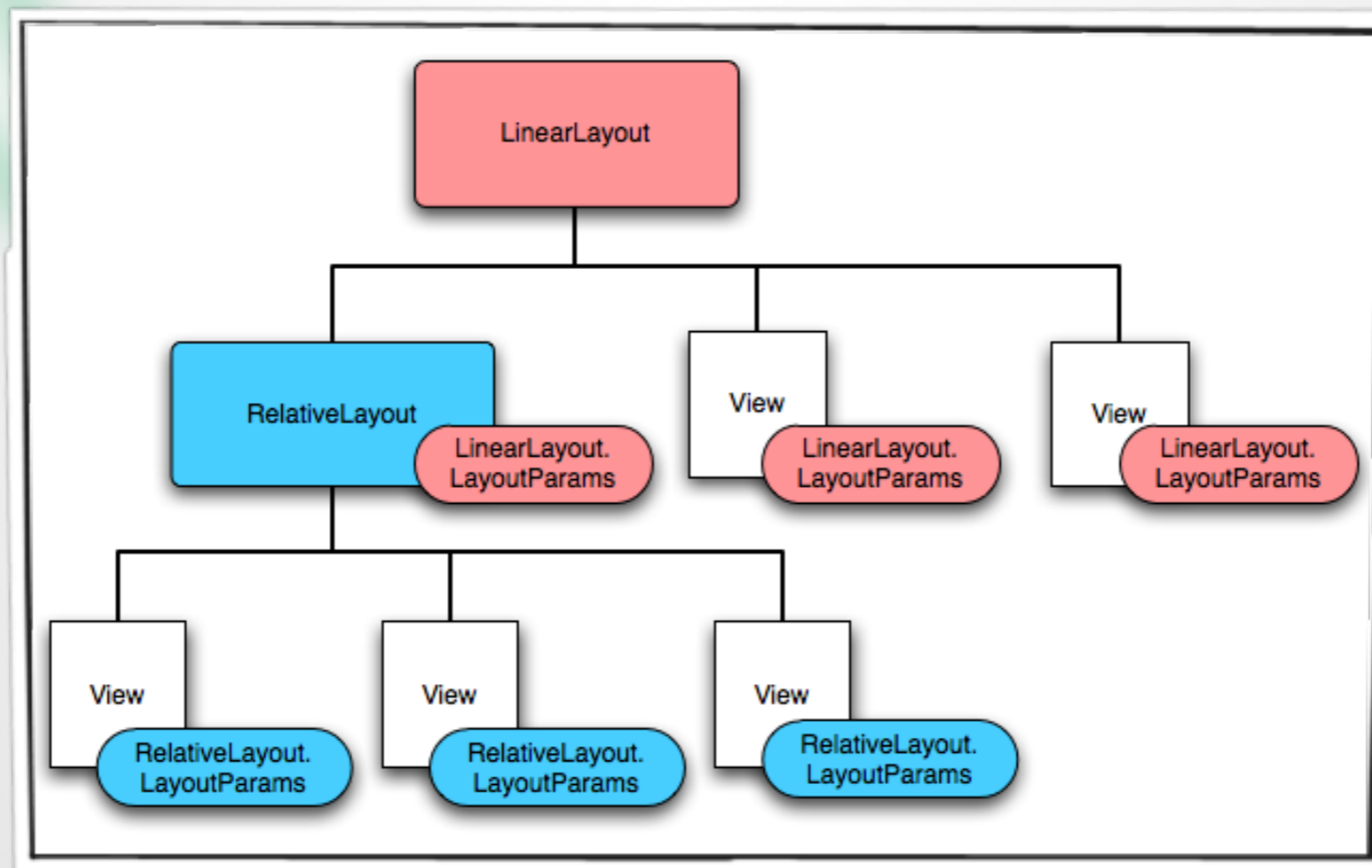
- An Activity in Android
  - Defined using a tree of view and viewgroup nodes
- setContentView() method
  - Called by the Activity to attach the tree to the screen for rendering





# LayoutParams

- Every viewgroup class uses a nested class that extends `ViewGroup.LayoutParams`
- Contains property types that defines the child's size and position
- Tells their parents how they want to be laid out



# Creating Layouts

Implementing a User Interface



Question:

- Benefits and drawbacks of Programmatic vs. Declarative UIs

# Programmatic UI Layout

- Programmatic UI Layout
  - Constructing and building the applications UI directly from source code
  - Disadvantage
    - small changes in layout can have a big effect on the source code

```
package com.android.hello;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

public class HelloAndroid extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        TextView tv = new TextView(this);
        tv.setText("Hello, Android");
        setContentView(tv);
    }
}
```

# Upgrading UI to XML Layout

- XML-based Layout
  - Inspired by web development model where the presentation of the application's UI is separated from the logic
  - Two files to edit
    - Java file - application logic
    - XML file - user interface

# Upgrading UI to XML Layout

The screenshot shows the Eclipse IDE interface for a Java project named 'UIExample'. The Package Explorer on the left shows the project structure, with 'UIExample.java' and 'main.xml' circled in red. The main editor displays the Java code for 'UIExample.java', with the 'onCreate' method circled in red. The Outline view on the right shows the class structure, and the Console at the bottom shows a DDMS log message.

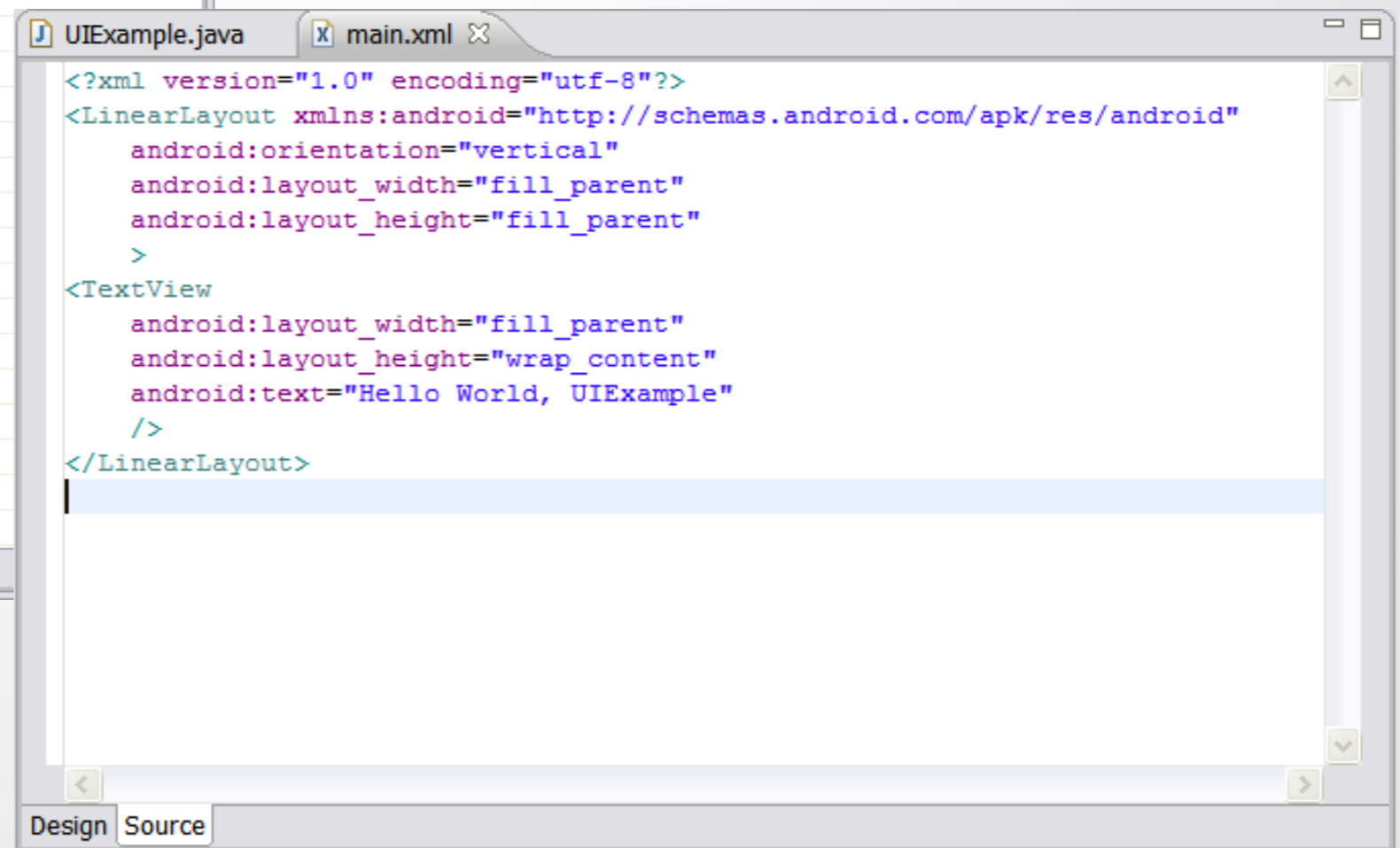
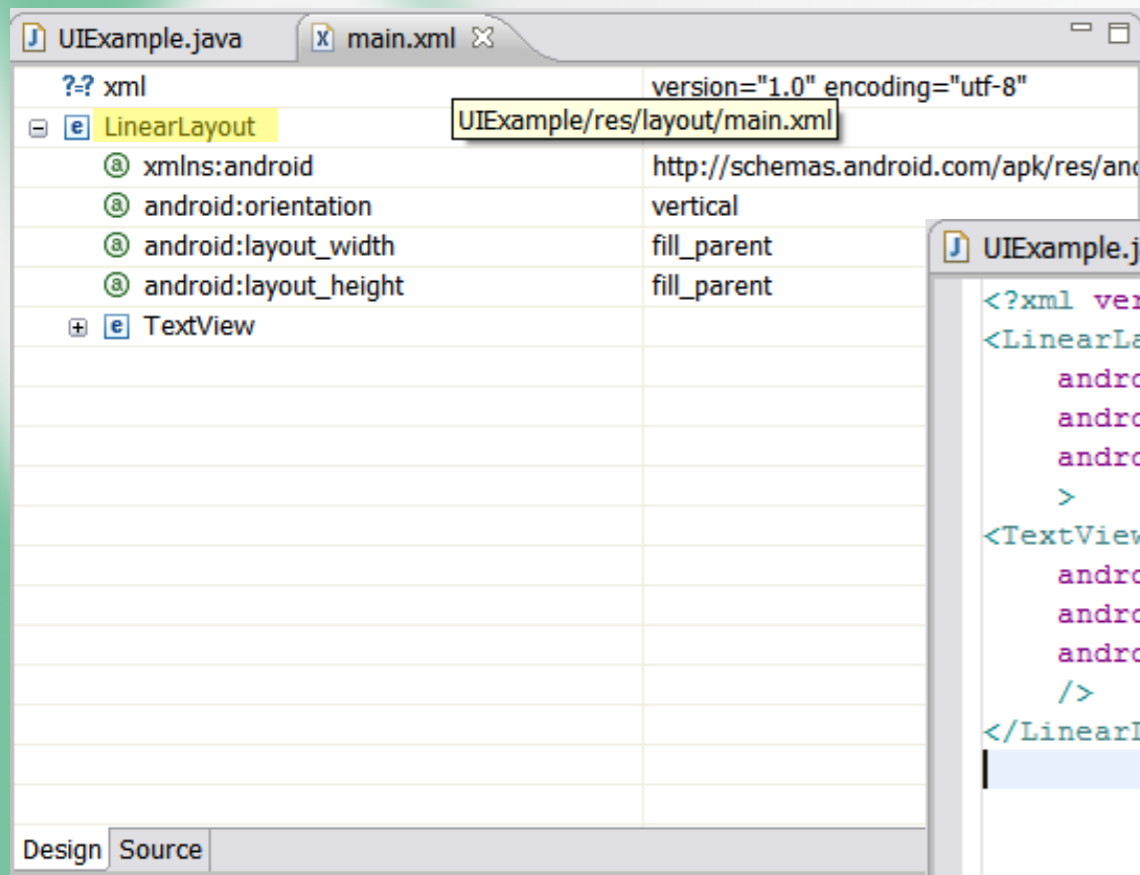
```
package pem.samplecode.ui;

import android.app.Activity;

public class UIExample extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

DDMS  
[2008-03-18 14:14:20 - adb] \* daemon not running. starting it now \*

# Upgrading UI to XML Layout





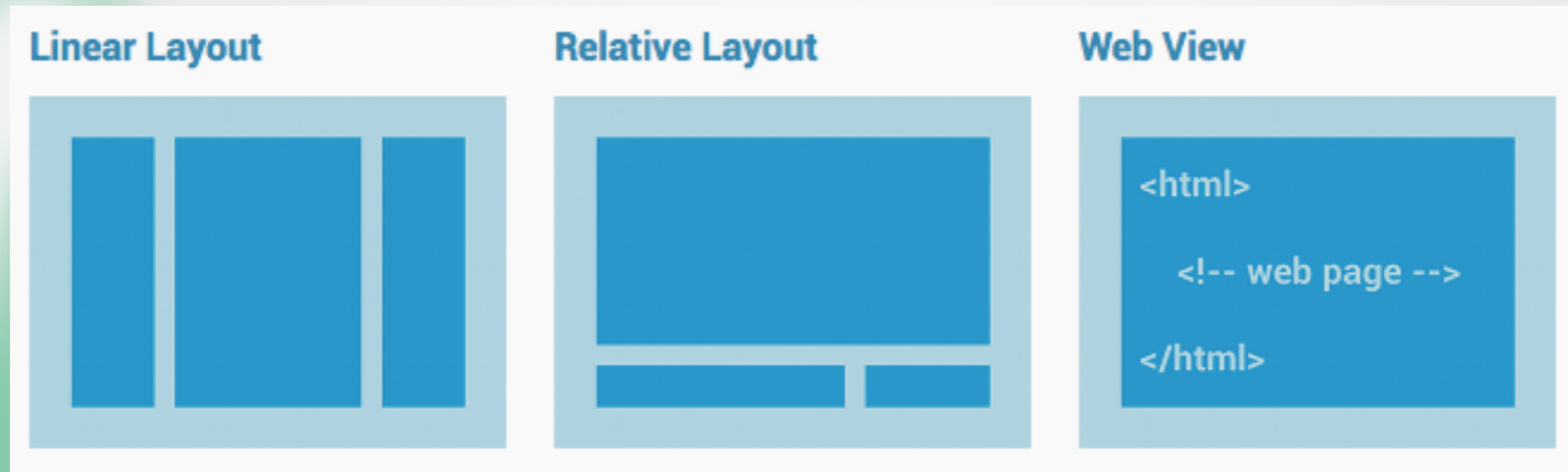


# Layouts

Implementing a User Interface



# Common Layouts

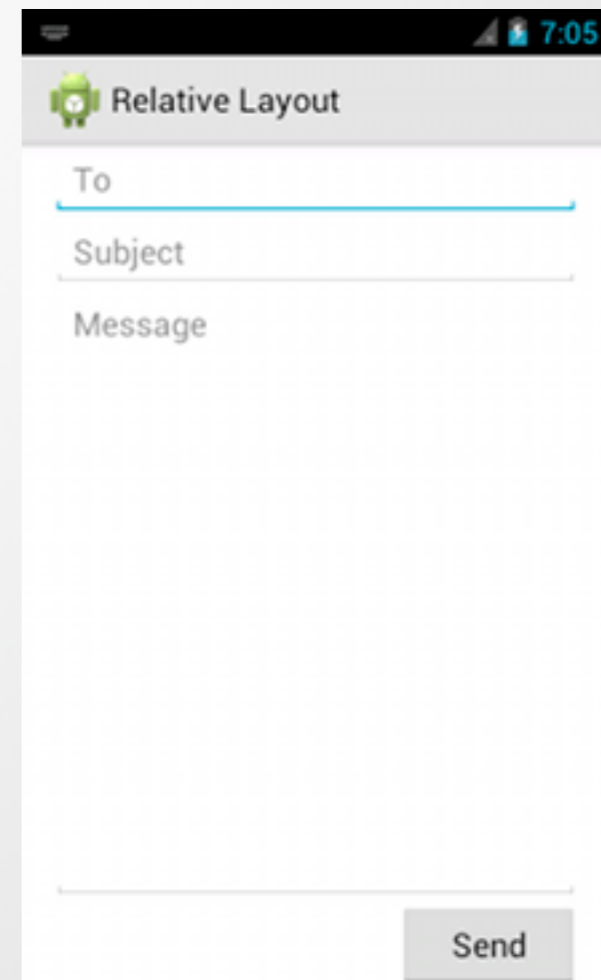


- Linear: Organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.
- Relative: Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent)
- Web View: Displays web pages.

# Linear Layout

- Aligns all children in a single direction, vertically or horizontally.
- Layout direction specified with the android:orientation attribute

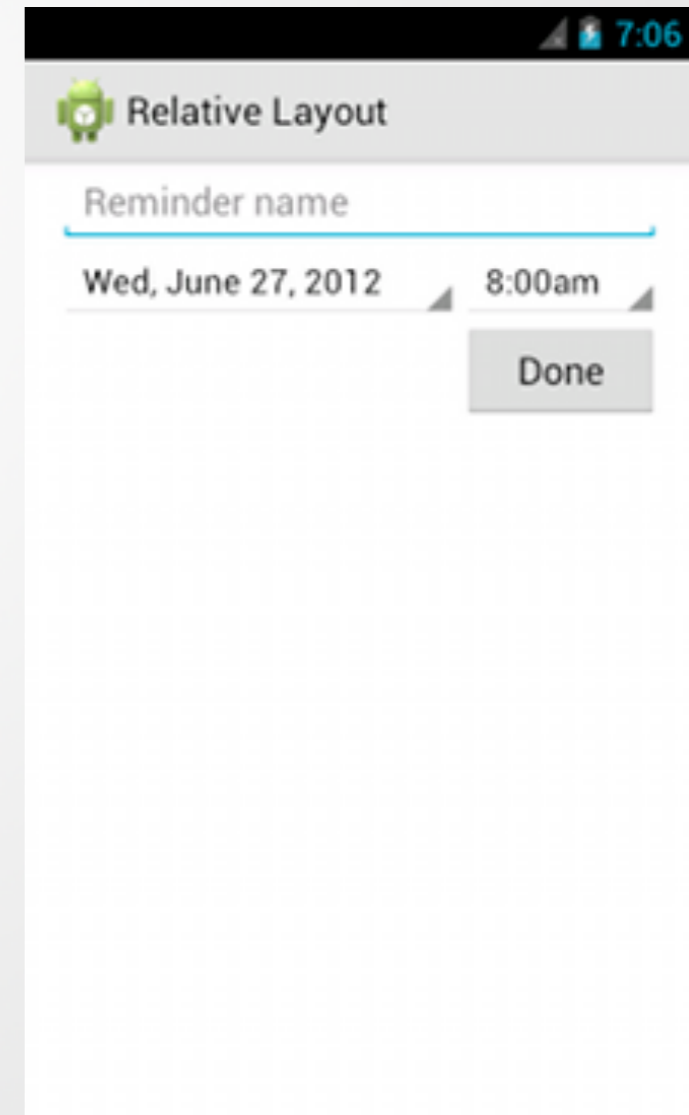
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
    <EditText
        android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />
</LinearLayout>
```



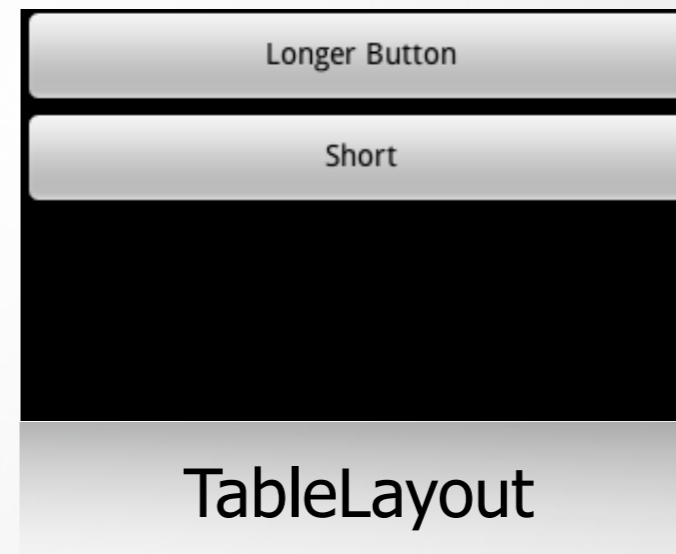
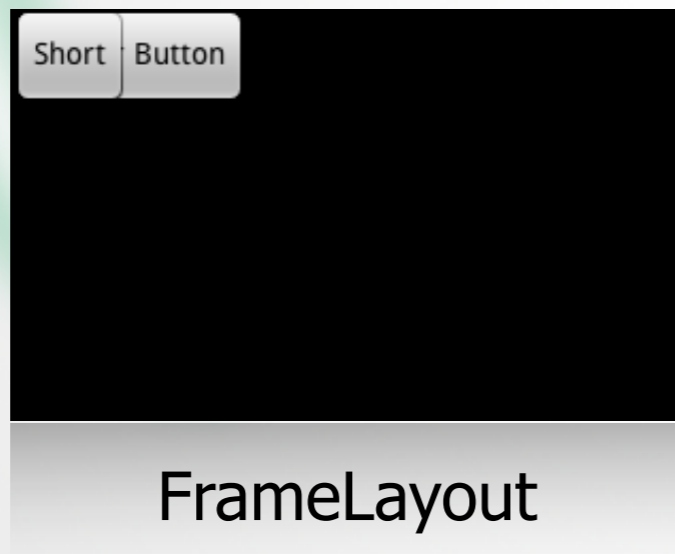
# RelativeLayout

- Lets children specify their position relative to each other (specified by ID), or to the parent.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp" >
    <EditText
        android:id="@+id/name"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:hint="@string/reminder" />
    <Spinner
        android:id="@+id/dates"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentLeft="true"
        android:layout_toLeftOf="@+id/times" />
    <Spinner
        android:id="@id/times"
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/name"
        android:layout_alignParentRight="true" />
    <Button
        android:layout_width="96dp"
        android:layout_height="wrap_content"
        android:layout_below="@id/times"
        android:layout_alignParentRight="true"
        android:text="@string/done" />
</RelativeLayout>
```



# Other Common Layout Objects

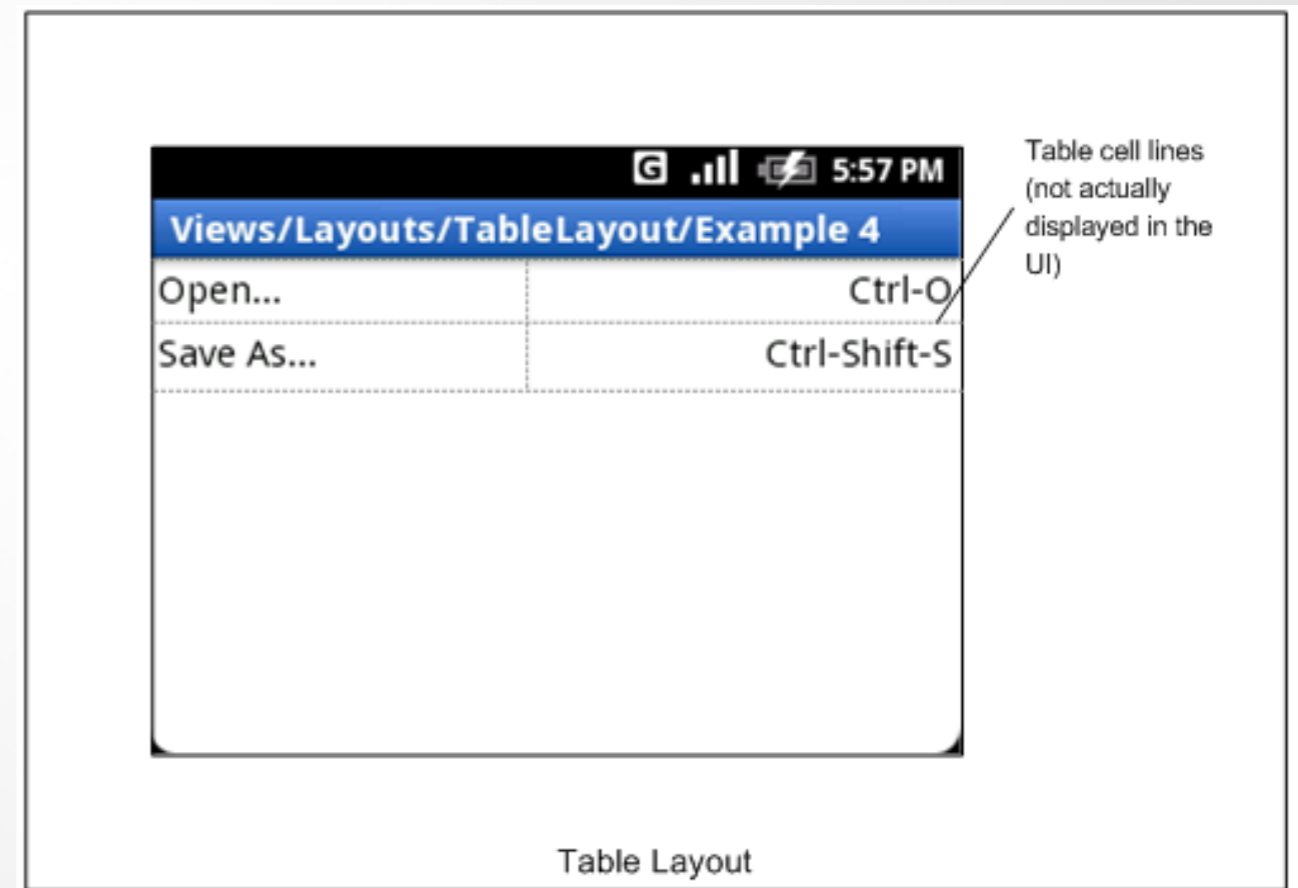


# FrameLayout

- Simplest layout object
- Intended as a blank reserved space on your screen that you can later fill with a single object
  - Example: a picture that you'll swap out
- All child elements are pinned to the top left corner of the screen
- Cannot specify a location for a child element

# TableLayout

- Positions its children into rows and columns
- Does not display border lines for their rows, columns, or cells
- Cells cannot span columns, as they can in HTML



# Important Layout Parameters

## Allgemein:

Layout-Height:	fill_parent, wrap_content, Pixels
Layout-Width:	fill_parent, wrap_content, Pixels
Id:	@+id/my_variable
Min-Height, Max-Height...	
Min-Width, Max-Width	

## Speziell:

EditText	Input type	text, textEmailAddress, number, numberDecimal
TextView, Button, EditText	Text	@string/resource_id
TextView	Text color, Text size	

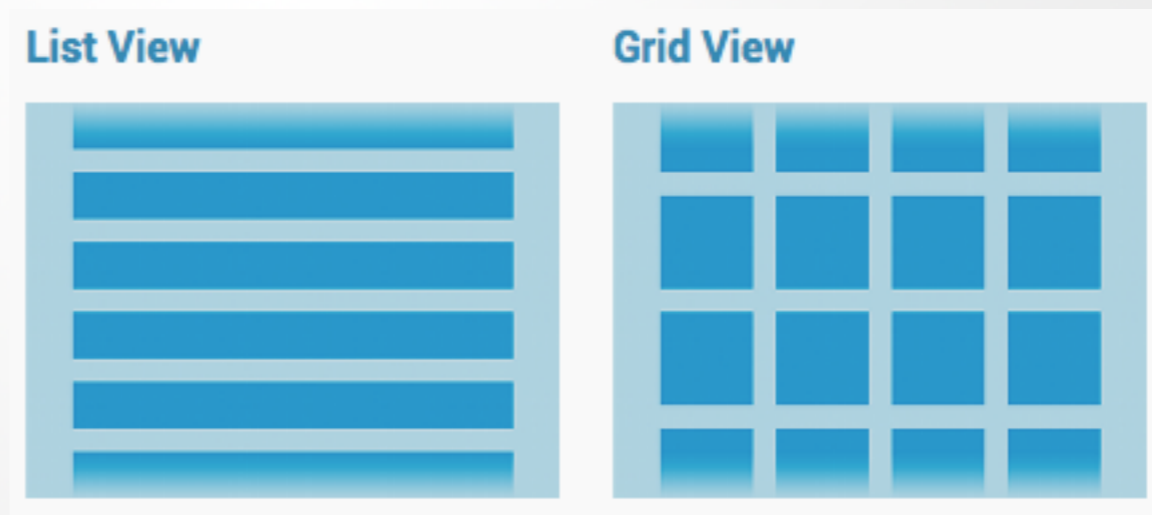


# Layouts with adapters

- Flexible layouts for dynamic content:

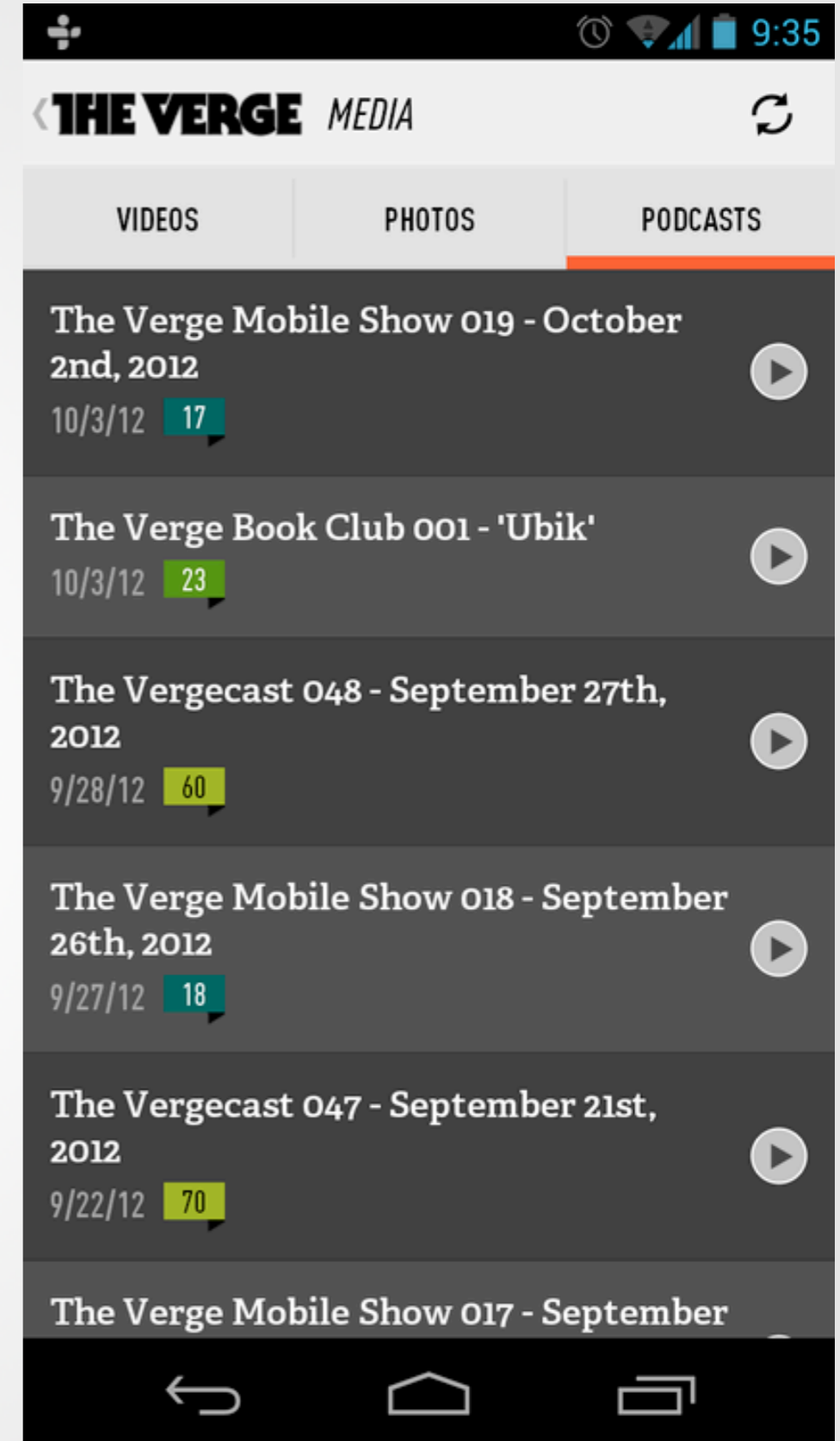
- List Views

- Grid View



# List View

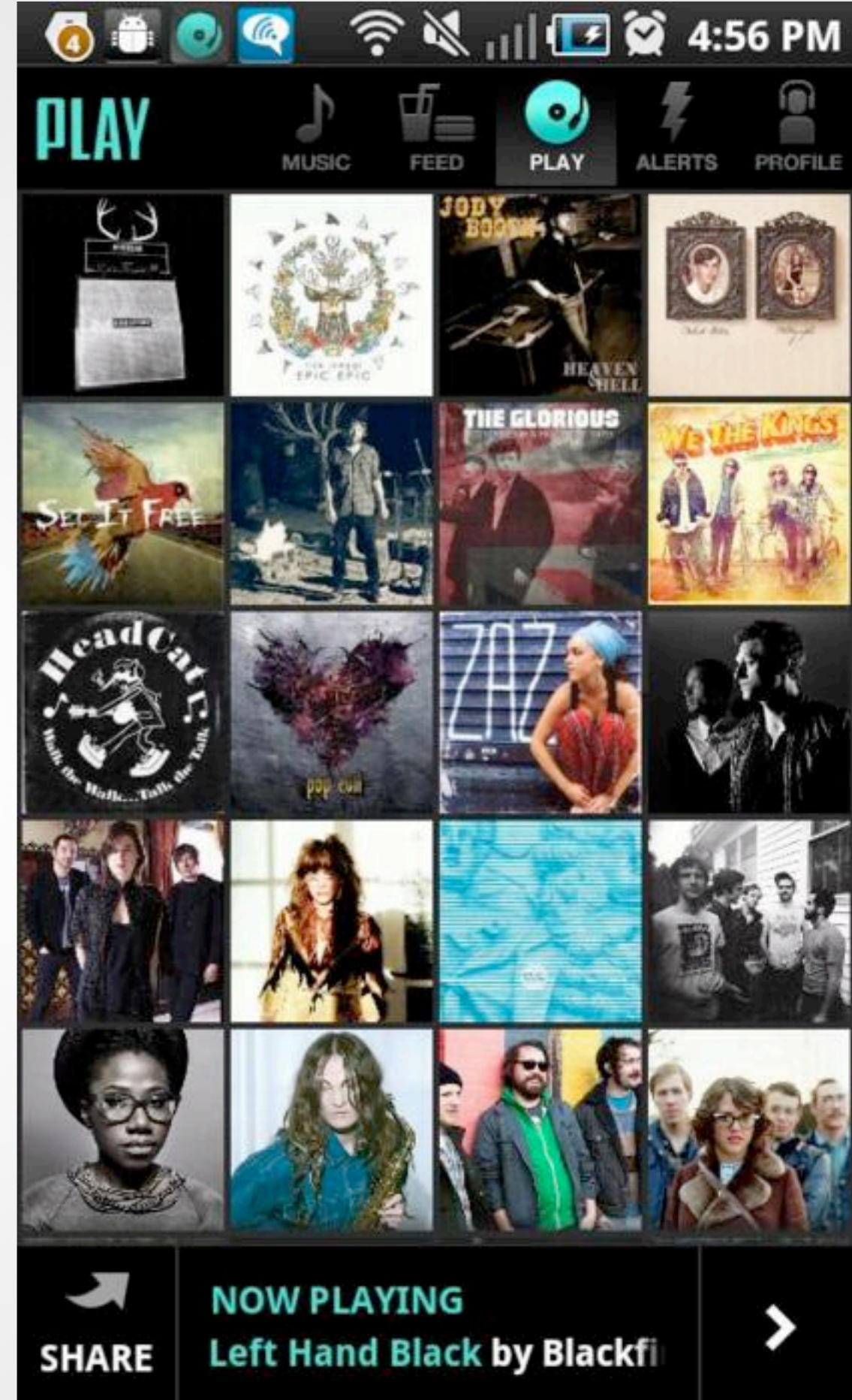
- View group that displays a list of scrollable items.
- List items are automatically inserted to the list using an Adapter.
- Adapters pulls content from a source (e.g. array or database query) and converts each item result into a view that's placed into the list.



<http://androidpttrns.com/tagged/list>

# Grid View

- GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.
- The grid items are automatically inserted to the layout using a ListAdapter.



# Grid View

## res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<GridView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/gridview"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:columnWidth="90dp"
    android:numColumns="auto_fit"
    android:verticalSpacing="10dp"
    android:horizontalSpacing="10dp"
    android:stretchMode="columnWidth"
    android:gravity="center"
/>
```

## HelloGridView.java

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    GridView gridView = (GridView) findViewById(R.id.gridview);
    gridView.setAdapter(new ImageAdapter(this));

    gridView.setOnItemClickListener(new OnItemClickListener() {
        public void onItemClick(AdapterView<?> parent,
            View v, int position, long id)
        {
            Toast.makeText>HelloGridView.this, "" +
                position, Toast.LENGTH_SHORT).show();
        }
    });
}
```

## ImageAdapter.java

```
public class ImageAdapter extends BaseAdapter {
    private Context mContext;

    public ImageAdapter(Context c) {
        mContext = c;
    }

    public int getCount() {
        return mThumbIds.length;
    }

    public Object getItem(int position) {
        return null;
    }

    public long getItemId(int position) {
        return 0;
    }

    // create a new ImageView for each item referenced by the Adapter
    public View getView(int position, View convertView, ViewGroup parent) {
        ImageView imageView;
        if (convertView == null) { // if not recycled, initialize attributes
            imageView = new ImageView(mContext);
            imageView.setLayoutParams(new GridView.LayoutParams(85, 85));
            imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
            imageView.setPadding(8, 8, 8, 8);
        } else {
            imageView = (ImageView) convertView;
        }

        imageView.setImageResource(mThumbIds[position]);
        return imageView;
    }

    // references to our images
    private Integer[] mThumbIds = {
        R.drawable.sample_2, R.drawable.sample_3,
        R.drawable.sample_4, R.drawable.sample_5,
        R.drawable.sample_6, R.drawable.sample_7,
        R.drawable.sample_0, R.drawable.sample_1,
        R.drawable.sample_2, R.drawable.sample_3,
        R.drawable.sample_4, R.drawable.sample_5,
        R.drawable.sample_6, R.drawable.sample_7,
        R.drawable.sample_0, R.drawable.sample_1,
        R.drawable.sample_2, R.drawable.sample_3,
        R.drawable.sample_4, R.drawable.sample_5,
        R.drawable.sample_6, R.drawable.sample_7
    };
}
```

# Online Reference

The screenshot shows the 'Hello, Views' tutorial page on the Android Developers website. The page is titled 'Hello, Views' and includes an introduction to the Views class. Below the text, there is a grid of 14 thumbnail images, each representing a different Android View widget. The thumbnails are arranged in two rows of seven. The first row includes: LinearLayout (a grid of colored squares), RelativeLayout (a form with a text field and buttons), TextView (a text field), DatePicker (a date picker dialog), TimePicker (a time picker dialog), Form Stuff (a form with various input fields), and Spinner (a spinner widget). The second row includes: ListView (a list of items), GridView (a grid of images), Gallery (a gallery of images), Toast (a toast message), MediaPlayer (a media player interface), and WebView (a web browser interface). The page also features a navigation menu on the left and a search bar at the top right.

<http://developer.android.com/guide/tutorials/views/index.html>

# UI Patterns

**Book** **Patterns** **Blog**

## Mobile Design Pattern Gallery:

UI Patterns for iOS, Android and More

By Theresa Neil  
O'Reilly Media, March 2012

**Buy Now**

### Chapters

- 1 **Navigation**
- 2 **Forms**
- 3 **Tables**
- 4 **Search, Sort & Filter**
- 5 **Tools**
- 6 **Charts**
- 7 **Invitations**
- 8 **Feedback & Affordance**
- 9 **Help**
- 10 **Anti-Patterns**

### Navigation

Includes patterns for Primary and Secondary Navigation

- 36 **Springboard**
- 13 **List Menu**
- 31 **Tab Menu**
- 15
- 3
- 5

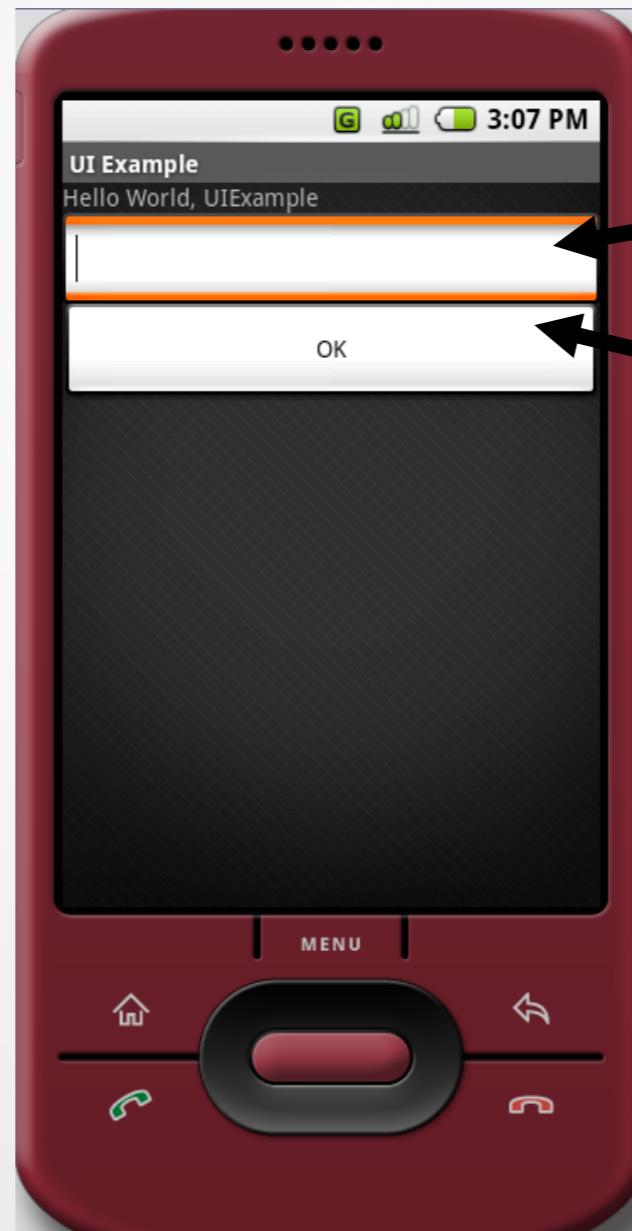
<http://www.mobiledesignpatterngallery.com/mobile-patterns.php>

# Hooking into a Screen Element

Implementing a User Interface



# Hooking into a Screen Element




**Text field**

**Button**



# Hooking into a Screen Element



```
UIExample.java main.xml X
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Hello World, UIExample"
        />

    <EditText
        android:id="@+id/name_entry"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        />

    <Button
        android:id="@+id/ok"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="OK"
        />

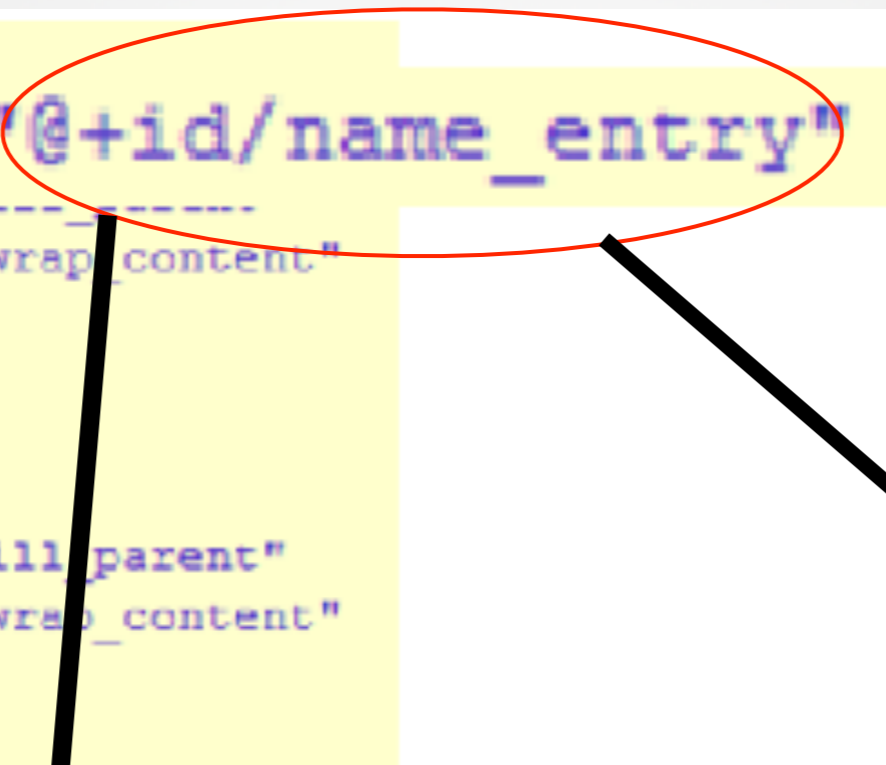
</LinearLayout>
```

Design Source

# Hooking into a Screen Element

```
<E
  android:id="@+id/name_entry"
  android:layout_height="wrap_content"
/>

<Button
  android:id="@+id/ok"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="OK"
/>
```

A diagram with two arrows. One arrow starts from the '@+id/name\_entry' part of the code and points down to the text '@+id syntax:'. The other arrow starts from the '@+id/ok' part of the code and points down to the text 'Any String value (no spaces)'.

**@+id** syntax:

Creates a resource number in the R class (R.java file) if one doesn't exist, or uses it if it does exist.

Any String value  
(no spaces)

# Hooking into a Screen Element

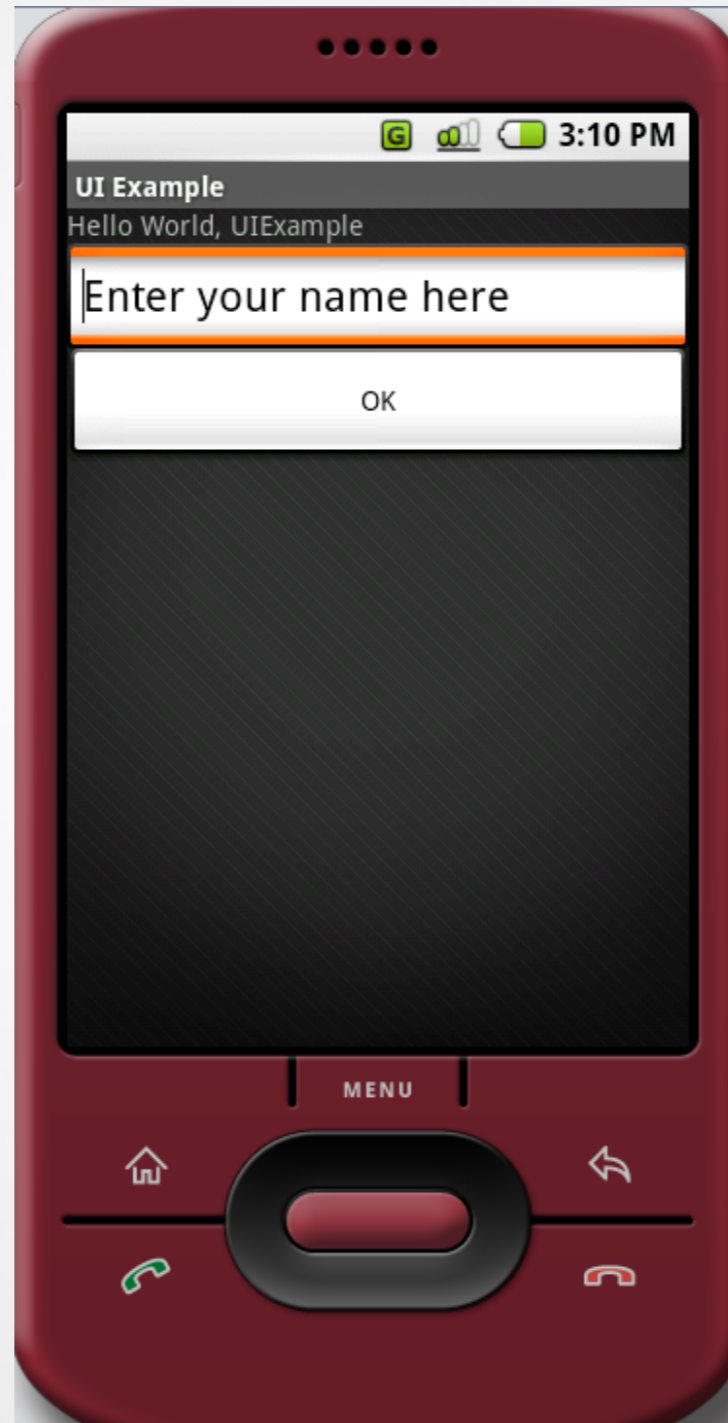
```
UIExample.java X main.xml
package pem.samplecode.ui;

import android.app.Activity;
import android.os.Bundle;
import android.widget.EditText;

public class UIExample extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //Add a handle to UI components
        EditText nameEntry = (EditText) findViewById(R.id.name_entry);
        nameEntry.setText("Enter your name here");
    }
}
```

# Hooking into a Screen Element



# Listening for UI Notifications

```
UIExample.java x main.xml
package pem.samplecode.ui;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class UIExample extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //Add a handle to UI components
        EditText nameEntry = (EditText)findViewById(R.id.name_entry);
        nameEntry.setText("Enter your name here");

        Button okButton = (Button)findViewById(R.id.ok);

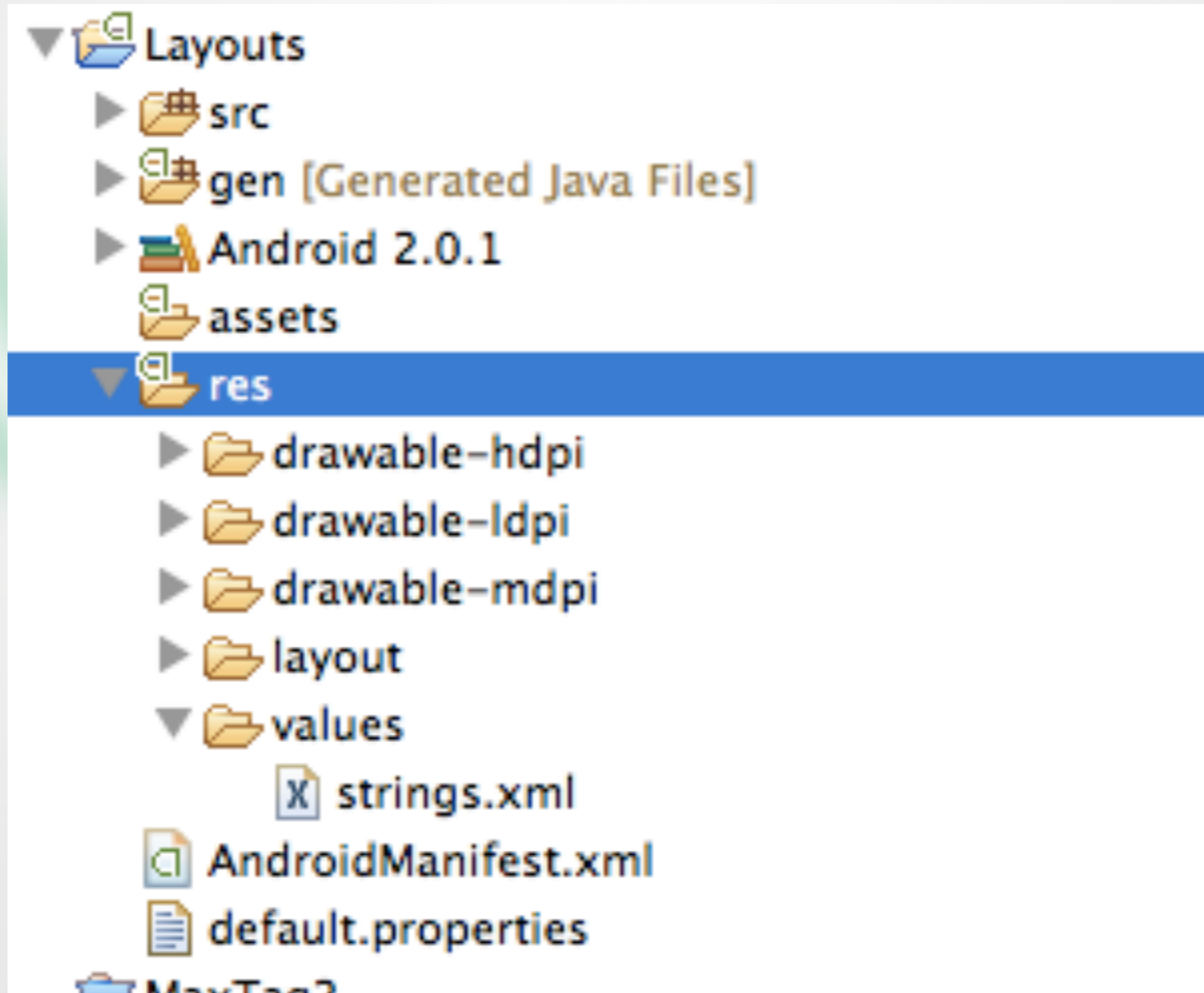
        //Create an anonymous class to act as a button click listener
        okButton.setOnClickListener(new OnClickListener()
        {
            public void onClick(View v){
                setResult(RESULT_OK, "Done!");
                finish();
            }
        });
    }
}
```

# Resource Folders and Localization

Implementing a User Interface

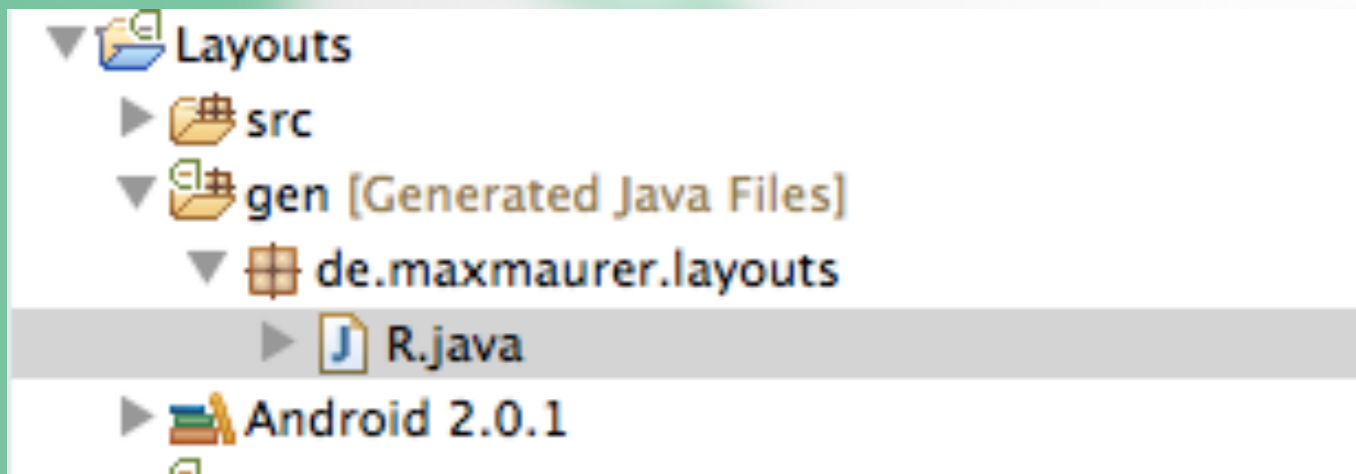


# Resource Folders



# Resource Folders

- Folder structure is automatically parsed into Resource-File
- Do not modify this file



```
⊕/* AUTO-GENERATED FILE. DO NOT MODIFY.⌵  
  
package de.maxmaurer.layouts;  
  
public final class R {  
    public static final class attr {  
    }  
    public static final class drawable {  
        public static final int icon=0x7f020000;  
    }  
    public static final class id {  
        public static final int Button01=0x7f050001;  
        public static final int Button02=0x7f050000;  
    }  
    public static final class layout {  
        public static final int main=0x7f030000;  
    }  
    public static final class string {  
        public static final int app_name=0x7f040001;  
        public static final int hello=0x7f040000;  
    }  
}
```



# Resource Folders

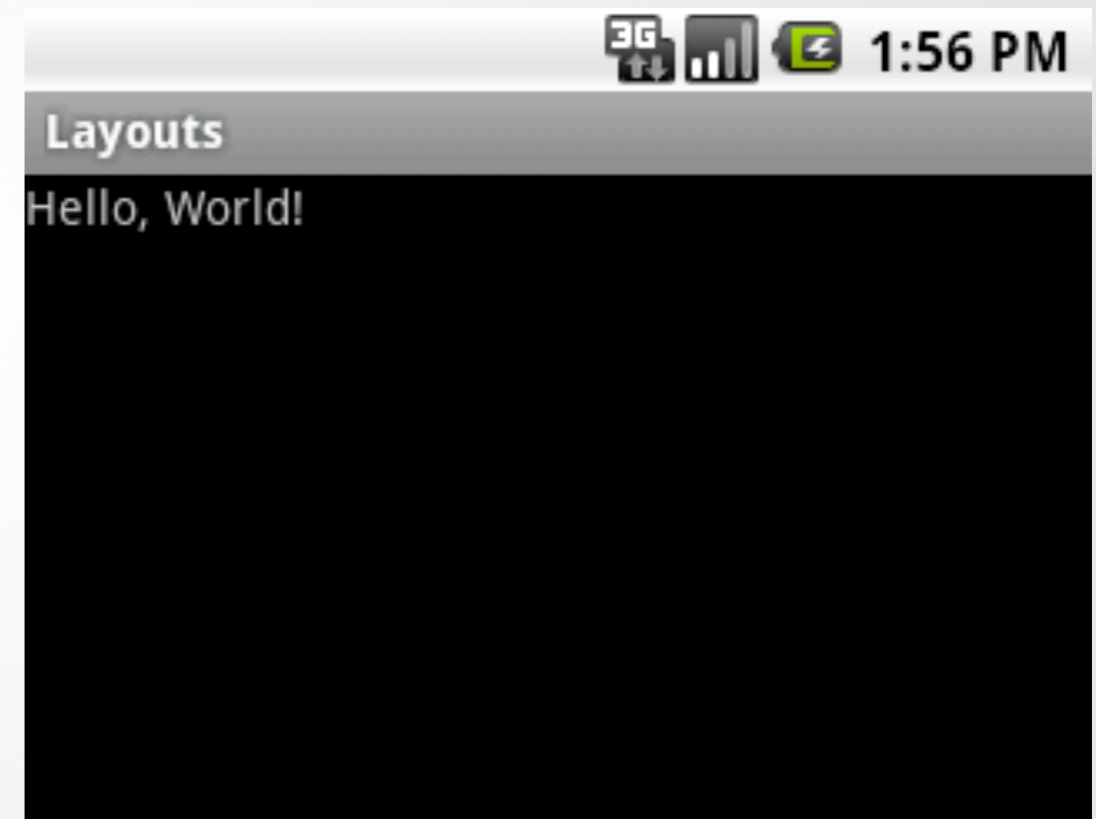
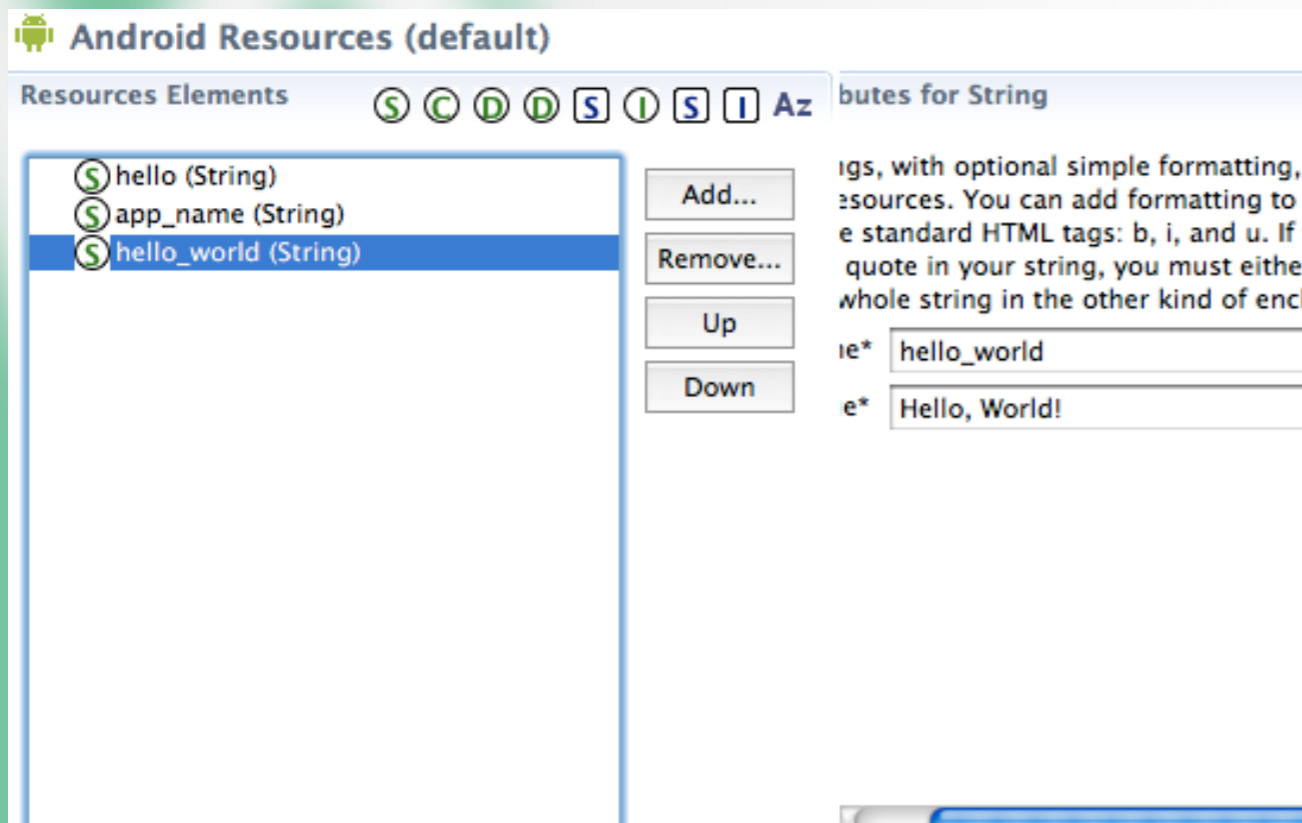
- Separate storage of Strings and Graphics
- Makes it easier to modify software parts
- Resources are accessed via „R.java“

```
package de.maxmaurer.layouts;

import android.app.Activity;

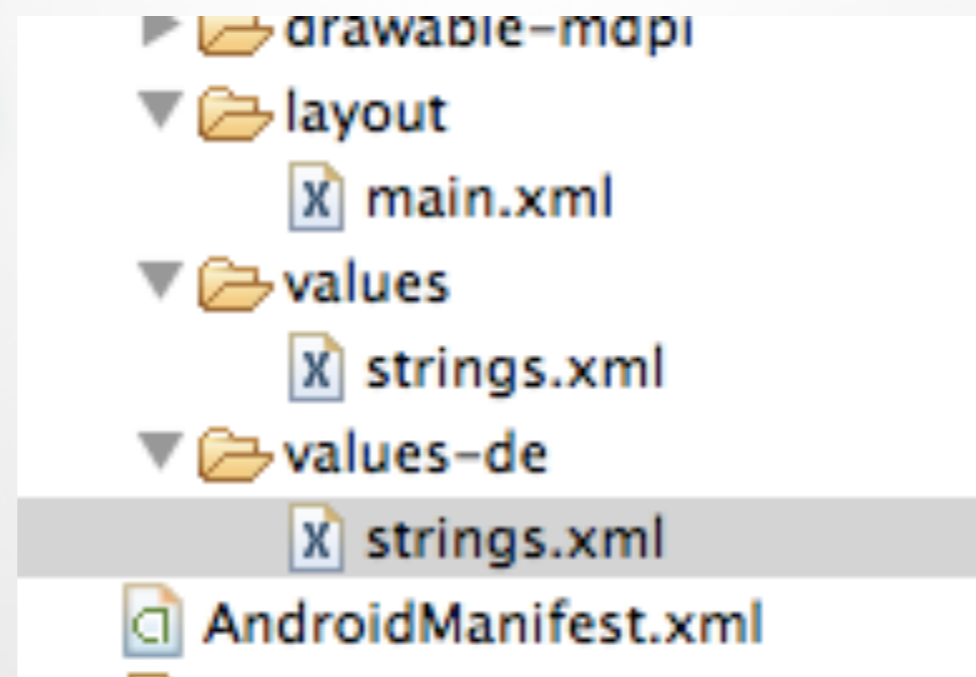
public class Layouts extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        View v = findViewById(R.layout.main);
        TextView tv = new TextView(this);
        tv.setText(getString(R.string.hello_world));
        setContentView(v);
    }
}
```

# Resource Folders

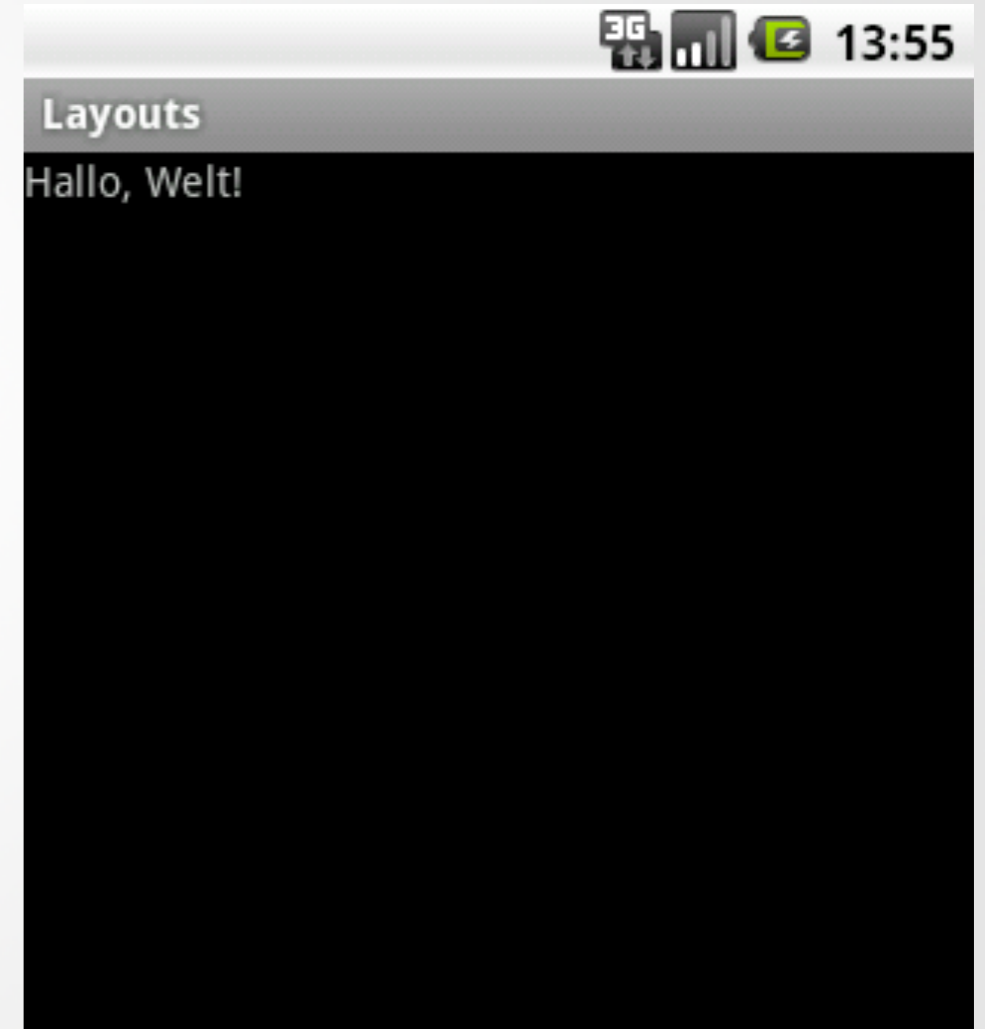
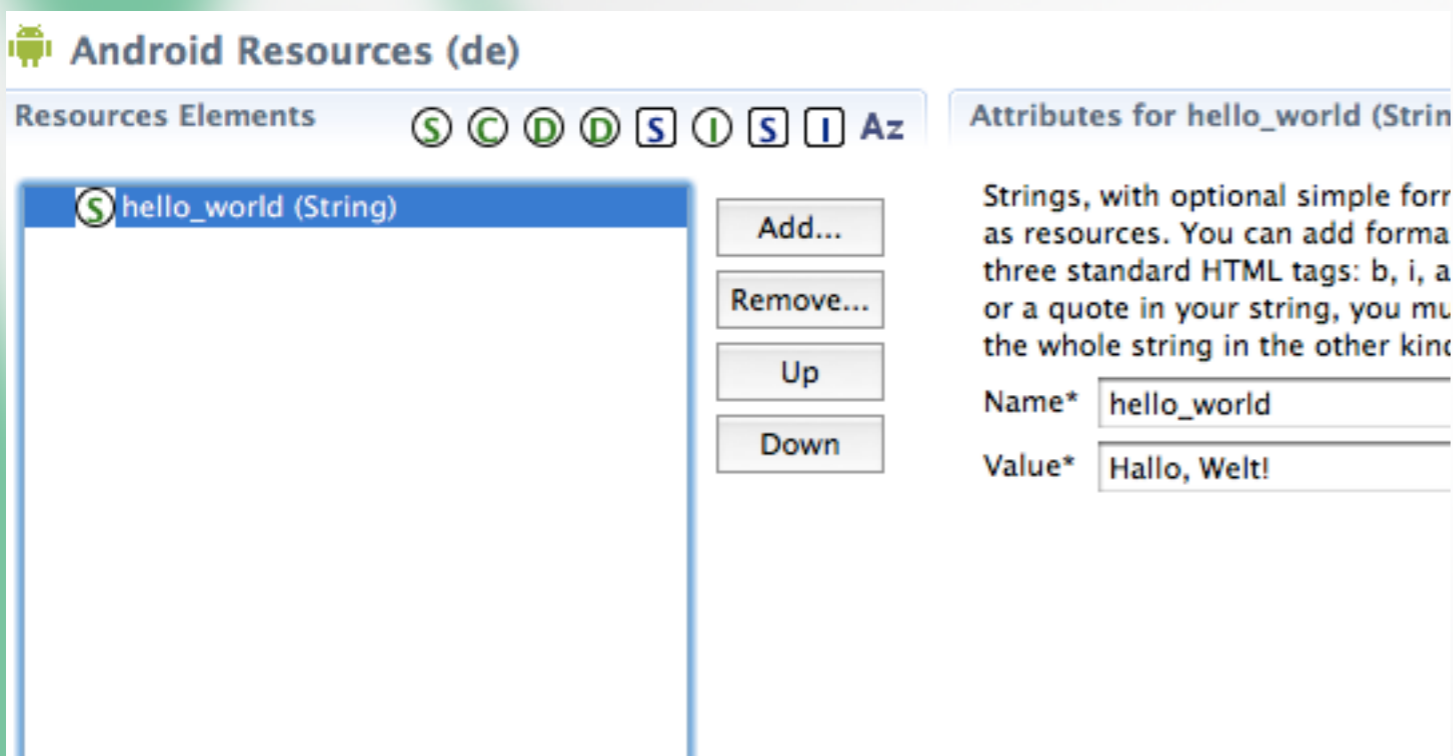


# Localization

- Creating folders for other languages does not need any code change
- Watch the application size!



# Localization



# Localization

- May be used for other device specific things as well:
  - Country
  - Screen dimensions
  - Screen orientation
  - Touchscreen type (finger, stylus)
  - and many more

```
MyApp/  
res/  
drawable-en-rUS-large-long-port-mdpi-finger-keysexposed-qwerty-navexposed-dpad-480x320/
```

# Application Themes

Implementing a User Interface



# Applying a Theme to Your Application

- Default theme: `android.R.style.Theme`
  - <http://developer.android.com/reference/android/R.style.html>
- Two ways to set the theme
  - Adding the theme attribute in `AndroidManifest.xml`
  - Calling `setTheme()` inside the `onCreate()` method

# Editing AndroidManifest.xml

- Adding the theme attribute in AndroidManifest.xml

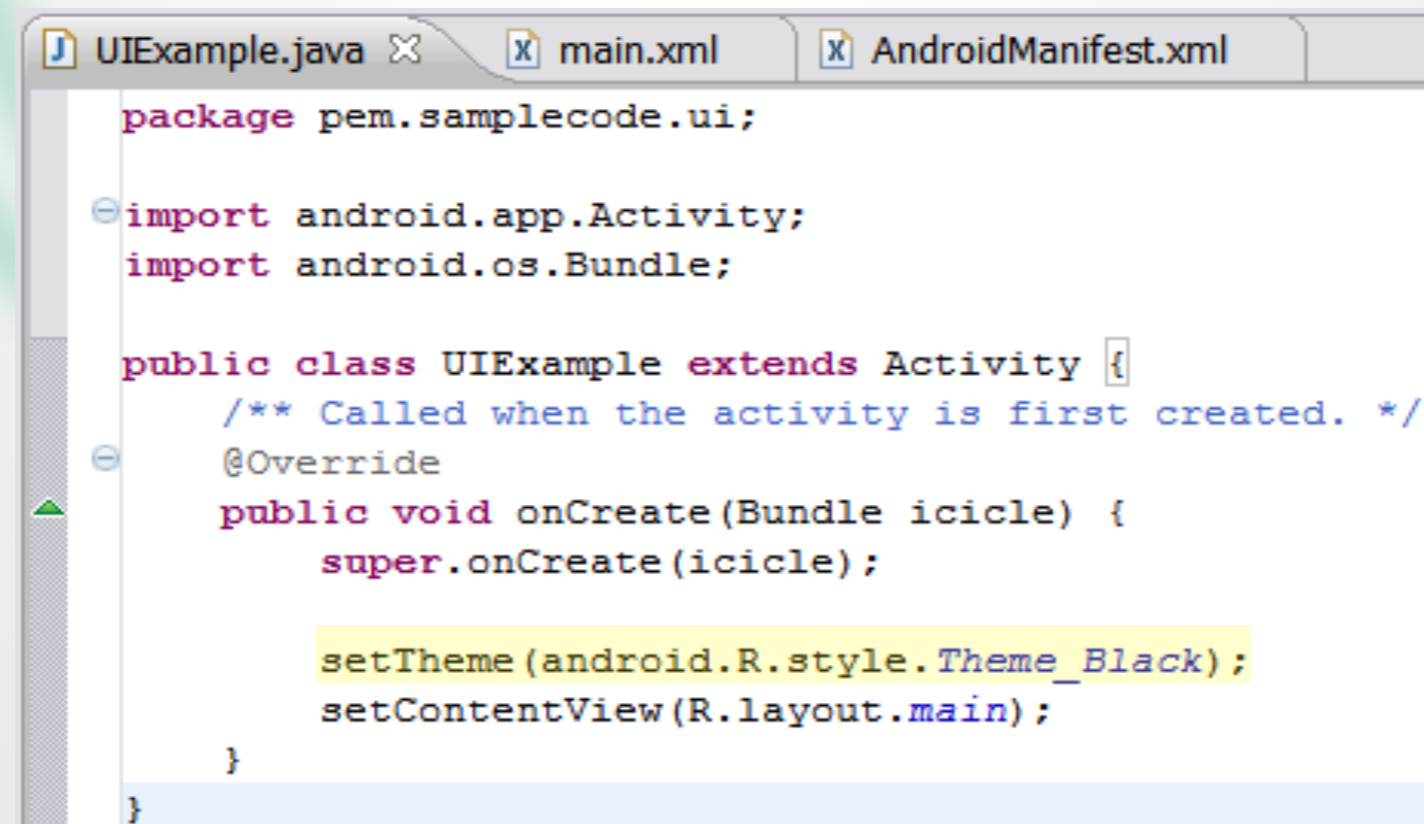


```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="pem.samplecode.ui">
    <application android:icon="@drawable/icon"
        android:theme="@android:style/Theme.Black">
        <activity android:name=".UIExample" android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```



# Applying a Theme using Code

- Calling `setTheme()` inside the `onCreate()` method



```
package pem.samplecode.ui;

import android.app.Activity;
import android.os.Bundle;

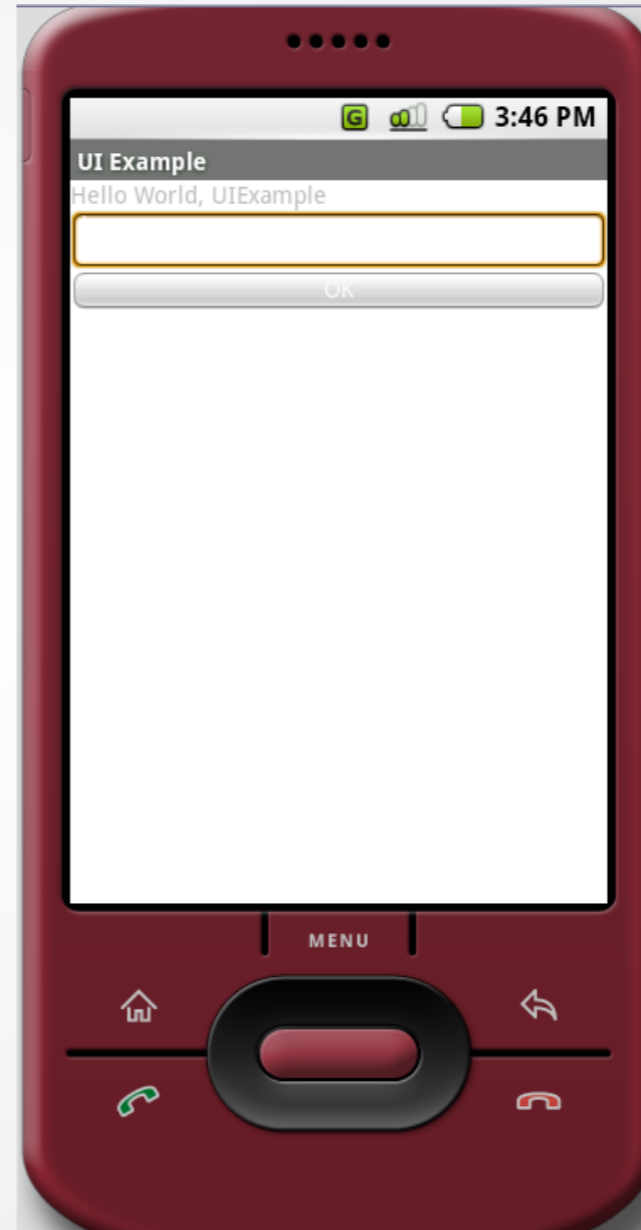
public class UIExample extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setTheme(android.R.style.Theme_Black);
        setContentView(R.layout.main);
    }
}
```

# Black



# Light White



# Themes are useful!

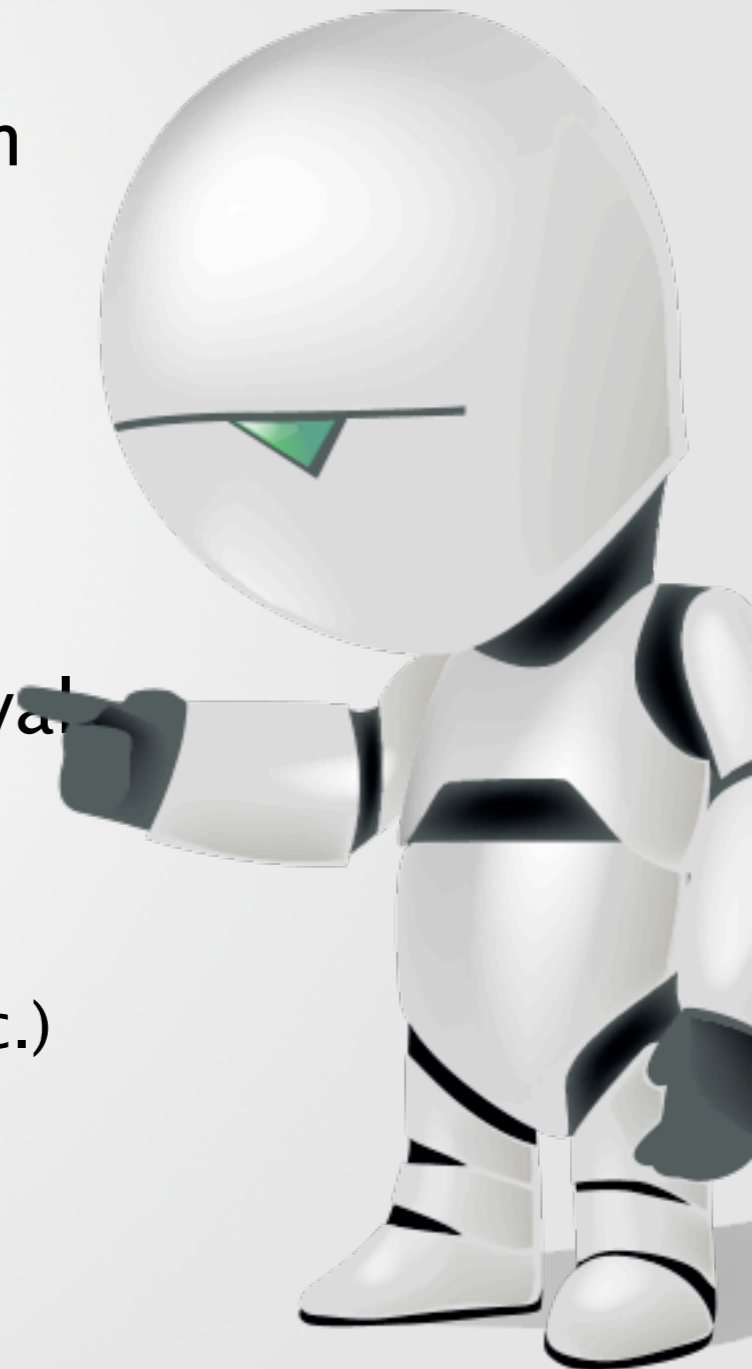


# Storing, Retrieving and Exposing Data in Android Apps



# Introduction

- All application data are private to an application
- Mechanisms to make data available for other applications
- Some simple/basic applications do not require information to be stored
- More elaborated software needs storage/retrieval functionality for different functionalities like:
  - Preserving an application's status (paused, first startup, etc.)
  - Saving user preferences (font size, sound on/off, etc.)
  - Working with complex data structures (calendars, maps, etc.)
  - ...



# Different Storage Methods

- Depending on the purpose of storing data, Android offers approaches with different complexity:
  - Store and retrieve simple name/value pairs
  - File operations (read, write, create, delete, etc.)
  - SQLite databases to work with complex data structures
  - Network operations to store and retrieve data from a network
  - Content providers to read/write data from an application's private data

**Preferences**

**File-IO**

**SQLite-Databases**

**Network Storage**

**Content-Providers**



**Preferences**

**File-IO**

**SQLite-Databases**

**Network Storage**

**Content-Providers**



**Preferences**

**File-IO**

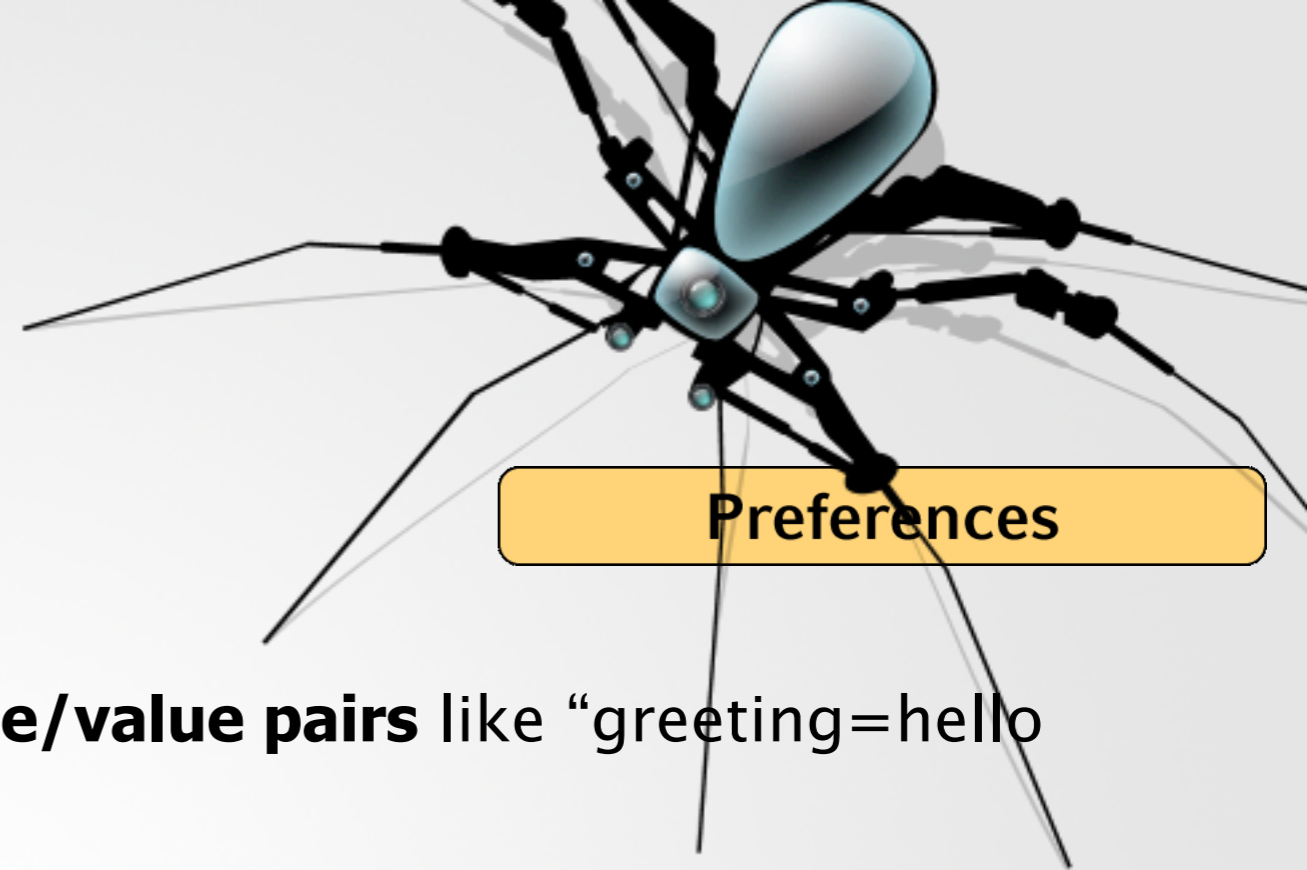
**SQLite-Databases**

**Network Storage**

**Content-Providers**



# Preferences



- Application preferences are simple **name/value pairs** like “greeting=hello name” or “sound = off”
- To work with preferences, Android offers an extremely simple approach
- Preferences can only be shared with other components in **the same** package
- Preferences cannot be shared across packages
- Private preferences will not be shared at all
- Storage location is not defined and inaccessible for other applications

**sound: off**

**username: hugo**

**font\_size: 10pt**

**pem: rocks**

# Using Preferences

## Preferences

- **Reading Preferences**

- `Context.getSharedPreferences(String name, int mode)` opens a set of preferences defined by “name”
- If a name is assigned, the preferences set will be shared amongst the components of the same package
- `Activity.getSharedPreferences(int mode)` can be used to open a set that is private to the calling activity

Opens a preferences set with the name “Preferences” in private mode

```
SharedPreferences settings = getSharedPreferences("Preferences", MODE_PRIVATE);  
boolean sound = settings.getBoolean("sound", false);
```

↑  
Reads a boolean parameter from the set. If the parameter does not exist, it will be created with the value defined in the second attribute. (other functions: `getAll()`, `getInt()`, `getString()`, etc.)

# Using Preferences

## Preferences

- **Writing** Preferences

- Changes on preferences are done using an Editor (SharedPreferences.Editor) object
- Each setting has one global Editor instance to administrate changes
- Consequence: each change will be available to every activity working with that preferences set

Gets the Editor instance of the preferences set

```
SharedPreferences.Editor editor = settings.edit();  
editor.putBoolean("sound", false);  
// COMMIT!!  
editor.commit();
```

Writes a boolean to a parameter

**Attention:** Changes are not drawn back to the settings before the commit is performed



**Preferences**

**File-IO**

**SQLite-Databases**

**Network Storage**

**Content-Providers**

# Files

- Files can be used to store bigger amounts of data than using preferences
- Android offers functionality to read/write files
- Only local files can be accessed
- **Advantage:** can store huge amounts of data
- **Disadvantage:** file update or changes in the format might result in huge programming effort

# Reading Files

- `Context.openFileInput(String name)` opens a `FileInputStream` of a private file associated with the application
- Throws a `FileNotFoundException` if the file doesn't exist

Open the file "test2.txt" (can be any name)

```
FileInputStream in = this.openFileInput("test2.txt");
```

```
...  
in.close();
```



Don't forget to close the InputStream at the end

# Writing Files

- `Context.openFileOutput(String name, int mode)` opens a `FileOutputStream` of a private file associated with the application
- If the file does not exist, it will be created
- `FileOutputStreams` can be opened in append mode, which means that new data will be added at the end of the file

Open the file "test2.txt" for writing (can be any name)



```
FileOutputStream out = this.openFileOutput("test2.txt", MODE_APPEND);  
...  
in.close();
```



Using `MODE-APPEND` opens the file in append mode

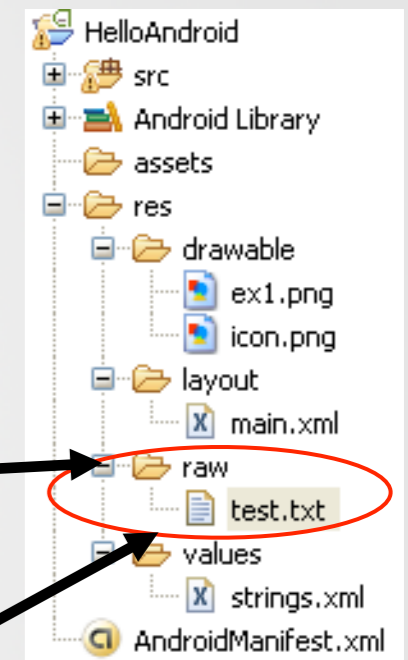


Don't forget to close the `InputStream` at the end

# Static Files

File-10

- To open static files packed in the application, use `Resources.openRawResource (R.raw.mydatafile)`
- The files have to be put in the folder `res/raw/`



Get the contexts resources

```
InputStream in = this.getResources().openRawResource(R.raw.test);  
...  
in.close();
```

↑  
Don't forget to close the InputStream at the end



# Using the SD-Card

File-IO

- Bigger amounts of data should usually be written/read from SD-Card
- Using the external storage requires permission
- Set it in Manifest.xml-File

```
<uses-permission  
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```



**Preferences**

**File-IO**

**SQLite-Databases**

**Network Storage**

**Content-Providers**

# SQLite Databases

- In some cases, files are not efficient
  - If multi-threaded data access is relevant
  - If the application is dealing with complex data structures that might change
  - Etc.
- Therefore, Android comes with built-in SQLite support
- Databases are private to the package that created them
- **Databases should not be used to store files**

# SQLite Databases

- SQLite is a lightweight software library
- Implements a fully ACID-compliant database
  - Atomicity
  - Consistency
  - Isolation
  - Durability
- Size only several kilobytes
- Some SQL statements are only partially supported (e.g. ALTER TABLE)
- Only few types of data
- See <http://www.sqlite.org/> for more information

# Creating a Database

SQLite-Databases

- Opening a database should create it when needed
- Creating a database always means taking care of future Versions
- Version-Numbers make sure which kind of DB is currently used
- An extra class usually called „DBAdapter.java“ is used for all database access

## SQLite-Databases

```
public class DBAdapter extends SQLiteOpenHelper {
    public static final String KEY_ROWID = "_id";
    private static final String TAG = "DBAdapter";

    private static final String DATABASE_NAME = "mydb";
    private static final String DATABASE_TABLE = "table_one";
    private static final int DATABASE_VERSION = 1;
    private static final String TABLE_CREATE = "create table "+DATABASE_TABLE+" (" +
        KEY_ROWID + " integer primary key autoincrement)";

    private SQLiteDatabase db;

    public DBAdapter(Context ctx) {
        super(ctx, DATABASE_NAME, null, DATABASE_VERSION);
        db=getWritableDatabase();
    }


    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(TABLE_CREATE);
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        Log.w(TAG, "Upgrading database from version " + oldVersion + " to "
            + newVersion + ", which will destroy all old data");
        db.execSQL("DROP TABLE IF EXISTS " + DATABASE_TABLE);
        onCreate(db);
    }
}
```

# Fetching Data

SQLite-Databases

- Data is provided using Cursors
- Cursors are the result of a specific query to the database holding the request result
- Cursors are traversed line by line
  - Similar to an Iterator in Java
- DBAdapter should provide request-methods that return such a Cursor

	id	someNumber
	1	8
	2	10
	3	2

# Fetching Data

To create a cursor, a query has to be executed either by SQL using `rawQuery()` or by more elaborated methods like `query()`



```
Cursor cur = dbase.rawQuery("SELECT * FROM test", null);
```

```
if (cur != null) {  
    int numColumn = cur.getColumnIndex("someNumber");  
    if (cur.moveToFirst()) {  
        do {  
            int num = cur.getInt(numColumn);  
            ...do something with it...  
        } while (cur.moveToNext());  
    }  
}
```

Attributes are retrieved with their index

Cursor offers different methods to retrieve different datatypes like `getInt(int index)` `getString(int index)` etc

`moveToNext()` moves the cursor to the next row. It returns false if no more row is available. Other possible moves are `moveToPrevious()` and `moveToFirst()`



# Fetching Data

```
public Cursor getAllEntrys() {  
    return db.query(DATABASE_TABLE, new String[] { KEY_ROWID }, null, null,  
        null, null, null);  
}
```

- query(), a more elaborated method
  - table: The table to query from
  - columns: Which columns to fetch
  - selection: the „Where“-Clause with placeholders?
  - selectionArgs: Values to fill placeholders
  - groupBy: SQL groupBy-Values
  - having: SQL having-Values
  - orderBy: How to order the resulting datasets

# Insert, Update

```
@Override  
public void onCreate(SQLiteDatabase db) {  
    db.execSQL(DATABASE_CREATE);  
}
```

```
db.execSQL("CREATE TABLE test (_id INTEGER PRIMARY KEY,  
someNumber INTEGER);");
```

## ● Some examples:

```
db.execSQL("Insert into test (_id, someNumber) values (1,8);");
```

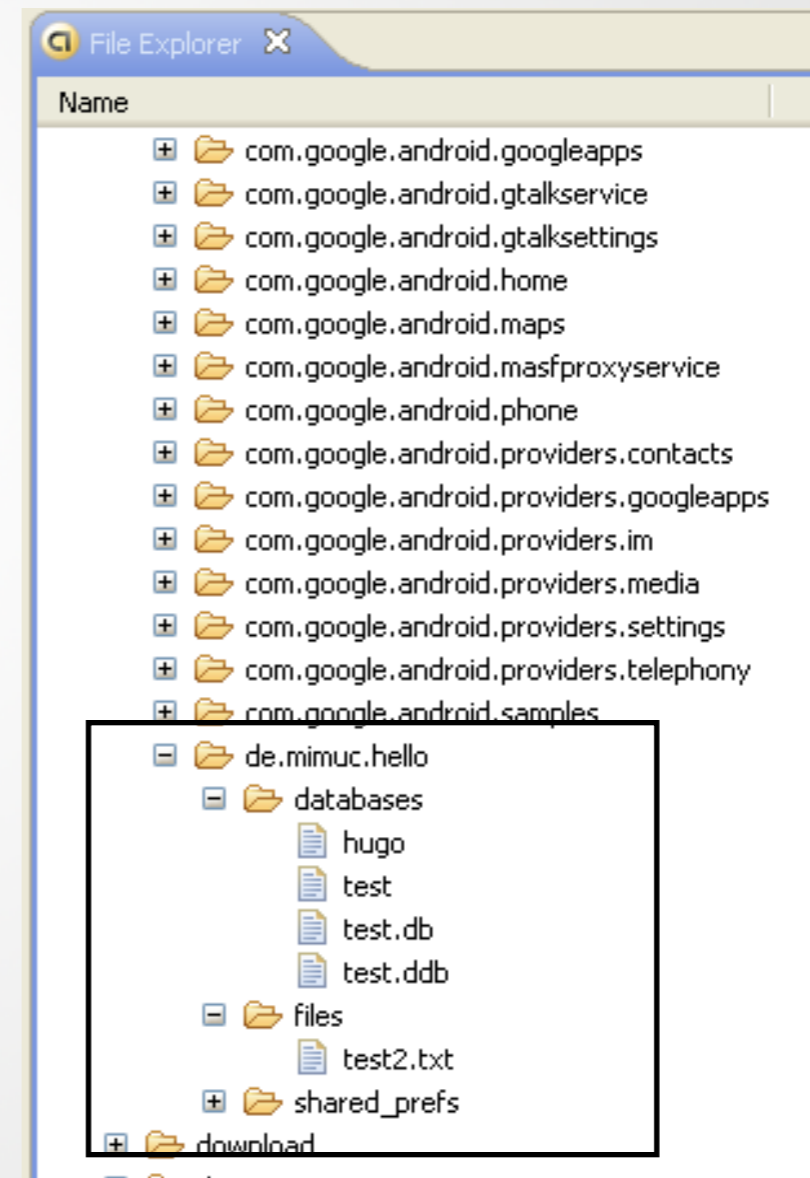
```
db.execSQL("DROP TABLE test");
```

# SQLiteQueryBuilder

- Optional interface to build correct SQL statements using code
- Usage:
  - Create new SQLiteQueryBuilder object
  - Then use setTables, appendWhere, appendColumns
  - In the end, use query or buildQuery

# Using the IDE to Check Files and Databases

- FileExplorer-View
- Check Files and Databases at / data/data/<package\_name>/ files|databases
- Only possible on a „rooted“ device/emulators.
- **Don't root the test devices!**





**Preferences**

**File-IO**

**SQLite-Databases**

**Network Storage**

**Content-Providers**

# Network Access

## Network Storage

- Android also supports network access to access files remotely (through the network)
- Two major packages:
  - `java.net.*` contains the standard Java network APIs
  - `android.net.*` adds additional helper classes to the standard Java APIs



**Preferences**

**File-IO**

**SQLite-Databases**

**Network Storage**

**Content-Providers**

# Content Providers

## Content-Providers

- All preferences, files and databases created by an Android application are private
- To share data with other applications, an application has to **create** a Content Provider
- To retrieve data of another application its content provider has to be **called**
- Androids **native Content Providers** include:
  - CallLog: information about placed and received calls
  - Settings.System: system settings and preferences



# Single-Exercise till 07.01.2013

Vocabulary Flashcard Application

<https://en.wikipedia.org/wiki/Flashcard>



# Flashcards

- 2 modes: set-up and quiz
- Set-up mode lets users input words in two languages.
- Quiz mode shows a word on screen and asks for the translation.
- Database stores words in two languages alongside the success rate in translating them.
- Words can be assigned to a category (Food, directions...)
- Answering a question brings up a new one.
- **This is no teamwork!** Make sure you work on your own. This is a multiple week exercise. So start early and enjoy Christmas!
- Submit till **07.01.2012 - 14:00** via UniWorX

Fragen?

